

Theoretische Informatik

Beweisideen 101

Kapitel 2

Lemma 2.5

Für jede Zahl $n \in \mathbb{N}$ existiert ein Wort $w_n \in (\Sigma_{bool})^n$, so dass

$$K(w_n) \geq |w_n| = n$$

d.h., es existiert für jede Zahl n ein nichtkomprimierbares Wort der Länge n .

Beweis:

Es gibt 2^n Wörter x_1, \dots, x_{2^n} über Σ_{bool} der Länge n . Wir bezeichnen $C(x_i)$ als den Bitstring des kürzesten Programms, der x_i generieren kann. Es ist klar, dass für $i \neq j : C(x_i) \neq C(x_j)$.

Die Anzahl der Bitstrings, i.e. der Wörter der Länge $< n$ über Σ_{bool} ist:

$$\sum_{i=1}^{n-1} 2^i = 2^n - 2 < 2^n$$

Also muss es unter den Wörtern x_1, \dots, x_{2^n} mindestens ein Wort x_k mit $K(x_k) \geq n$ geben. ■

Satz 2.2

Sei L eine Sprache über Σ_{bool} . Sei, für jedes $n \in \mathbb{N} \setminus \{0\}$, z_n das n -te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, dass das Entscheidungsproblem (Σ_{bool}, L) löst, dann gilt für alle $n \in \mathbb{N} \setminus \{0\}$, dass

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

wobei c eine von n unabhängige Konstante ist.

Beweisidee:

Wir können aus A_L , ein Programm entwerfen, dass das kanonisch n -te Wort generiert, indem wir in der kanonischen Reihenfolge alle Wörter $x \in (\Sigma_{bool})^*$ durchgehen und mit A_L entscheiden, ob $x \in L$. Dann können wir einen Counter c haben und den Prozess abbrechen, wenn der Counter $c = n$ wird und dann dieses Wort ausgeben.

Wir sehen, dass dieses Programm ausser der Eingabe n immer gleich ist. Sei die Länge dieses Programms c , dann können wir für das n -te Wort der Sprache L , z_n , die Kolmogorov-Komplexität auf n reduzieren, bzw:

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

■

Lemma 2.6

Sei n_1, n_2, n_3, \dots eine steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$. Für jedes $i \in \mathbb{N} \setminus \{0\}$ sei q_i die grösste Primzahl, die die Zahl n_i teilt. Dann ist die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ unendlich.

Beweis: Wir beweisen diese Aussage per Widerspruch:

Nehmen wir zum Widerspruch an, dass die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ sei endlich. Sei q_m die grösste Primzahl in Q . Dann können wir jede Zahl n_i eindeutig als

$$n_i = q_1^{r_{i,1}} \cdot q_2^{r_{i,2}} \cdot \dots \cdot q_m^{r_{i,m}}$$

für irgendwelche $r_{i,1}, r_{i,2}, \dots, r_{i,m} \in \mathbb{N}$ darstellen. Sei c die binäre Länge eines Programms, dass diese $r_{i,j}$ als Eingaben nimmt und n_i erzeugt (A ist für alle $i \in \mathbb{N}$ bis auf die Eingaben $r_{i,1}, \dots, r_{i,m}$ gleich).

Dann gilt:

$$K(n_i) \leq c + 8 \cdot (\lceil \log_2(r_{i,1} + 1) \rceil + \lceil \log_2(r_{i,2} + 1) \rceil + \dots + \lceil \log_2(r_{i,m} + 1) \rceil)$$

Die multiplikative Konstante 8 kommt daher, dass wir für die Zahlen $r_{i,1}, r_{i,2}, \dots, r_{i,m}$ dieselbe Kodierung, wie für den Rest des Programmes verwenden (z.B. ASCII-Kodierung), damit ihre Darstellungen eindeutig voneinander getrennt werden können. Weil $r_{i,j} \leq \log_2 n_i, \forall j \in \{1, \dots, m\}$ erhalten wir

$$K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil, \forall i \in \mathbb{N} \setminus \{0\}$$

Weil m und c Konstanten unabhängig von i sind, kann

$$\lceil \log_2 n_i \rceil / 2 \leq K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

$$\lceil \log_2 n_i \rceil / 2 \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

nur für endlich viele $i \in \mathbb{N} \setminus \{0\}$ gelten.

Dies ist ein Widerspruch!

Folglich ist die Menge Q unendlich.

■

Kapitel 3

Lemma 3.3

Sei $A = (Q, \Sigma, \delta_A, q_0, F)$ ein EA. Seien $x, y \in \Sigma^*$, $x \neq y$, so dass

$$\hat{\delta}_A(q_0, x) = p = \hat{\delta}_A(q_0, y)$$

für ein $p \in Q$ (also $x, y \in \text{Kl}[p]$). Dann existiert für jedes $z \in \Sigma^*$ ein $r \in Q$, so dass xz und $yz \in \text{Kl}[r]$, also gilt insbesondere

$$xz \in L(A) \iff yz \in L(A)$$

Beweis:

Aus der Existenz der Berechnungen

$(q_0, x) \mid_A^* (p, \lambda)$ und $(q_0, y) \mid_A^* (p, \lambda)$ von A folgt die Existenz der Berechnungen auf xz und yz :

$(q_0, xz) \mid_A^* (p, z)$ und $(q_0, yz) \mid_A^* (p, z)$ für alle $z \in \Sigma^*$.

Wenn $r = \hat{\delta}_A(p, z)$ ist, dann ist die Berechnung von A auf xz und yz :

$(q_0, xz) \mid_A^* (p, z) \mid_A^* (r, \lambda)$ und $(q_0, yz) \mid_A^* (p, z) \mid_A^* (r, \lambda)$.

Wenn $r \in F$, dann sind beide Wörter xz und yz in $L(A)$. Falls $r \notin F$, dann sind $xz, yz \notin L(A)$. ■

Lemma 3.4: Pumping-Lemma

Sei L regulär. Dann existiert eine Konstante $n_0 \in \mathbb{N}$, so dass sich jedes Wort $w \in \Sigma^*$ mit $|w| \geq n_0$ in drei Teile y, x und z zerlegen lässt, das heisst $w = yxz$, wobei

- (i) $|yx| \leq n_0$,
- (ii) $|x| \geq 1$ und
- (iii) entweder $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^kz \mid k \in \mathbb{N}\} \cap L = \emptyset$.

Beweis:

Sei $L \in \Sigma^*$ regulär. Dann existiert ein EA $A = (Q, \Sigma, \delta_A, q_0, F)$, so dass $L(A) = L$.

Sei $n_0 = |Q|$ und $w \in \Sigma^*$ mit $|w| \geq n_0$. Dann ist $w = w_1w_2\dots w_{n_0}u$, wobei $w_i \in \Sigma$ für $i = 1, \dots, n_0$ und $u \in \Sigma^*$. Betrachten wir die Berechnung auf $w_1w_2\dots w_{n_0}$:

$$(q_0, w_1w_2w_3\dots w_{n_0}) \mid_A (q_1, w_2w_3\dots w_{n_0}) \mid_A (q_2, w_3\dots w_{n_0}) \mid_A \dots \mid_A (q_{n_0-1}, w_{n_0}) \mid_A (q_{n_0}, \lambda)$$

In dieser Berechnung kommen n_0+1 Zustände q_0, q_1, \dots, q_{n_0} vor. Da $|Q| = n_0$, existieren $i, j \in \{0, 1, \dots, n_0\}, i < j$, so dass $q_i = q_j$. Daher haben wir in der Berechnung die Konfigurationen

$$(q_0, w_1 w_2 w_3 \dots w_{n_0}) \mid_A^* (q_i, w_{i+1} w_{i+2} \dots w_{n_0}) \mid_A^* (q_i, w_{j+1} \dots w_{n_0}) \mid_A^* (q_{n_0}, \lambda)$$

Dies impliziert

$$(q_i, w_{i+1} w_{i+2} \dots w_j) \mid_A^* (q_i, \lambda) \quad (1)$$

Wir setzen nun $y = w_1 \dots w_i$, $x = w_{i+1} \dots w_j$ und $z = w_{j+1} \dots w_{n_0}$, so dass $w = yxz$.

Wir überprüfen nun die Eigenschaften (i), (ii) und (iii):

- (i) $yx = w_1 \dots w_i w_{i+1} \dots w_j$ und daher $|yx| = j \leq n_0$.
- (ii) Da $|x| \geq j - i$ und $i < j$, ist $|x| \geq 1$.
- (iii) (1) impliziert $(q_i, x^k) \mid_A^* (q_i, \lambda)$ für alle $k \in \mathbb{N}$. Folglich gilt für alle $k \in \mathbb{N}$:

$$(q_0, yx^k z) \mid_A^* (q_i, x^k z) \mid_A^* (q_i, z) \mid_A^* (\hat{\delta}_A(q_i, z), \lambda)$$

Wir sehen, dass für alle $k \in \mathbb{N}$ die Berechnungen im gleichen Zustand $q_{end} = \hat{\delta}_A(q_i, z)$ enden. Falls also $q_{end} \in F$, akzeptiert A alle Wörter aus $\{yx^k z \mid k \in \mathbb{N}\}$. Falls $q_{end} \notin F$, dann akzeptiert A kein Wort aus $\{yx^k z \mid k \in \mathbb{N}\}$.

■

Lemma 3.6

Sei $L_k = \{x1y \mid x \in (\Sigma_{bool})^*, y \in (\Sigma_{bool})^k\}$.

Für alle $k \in \mathbb{N} \setminus \{0\}$ muss jeder EA, der L_k akzeptiert, mindestens 2^k Zustände haben.

Beweis:

Sei $B_k = (Q_k, \Sigma_{bool}, \delta_k, q_{0k}, F_k)$ ein EA mit $L(B_k) = L_k$.

Nach **Lemma 3.3** gilt für $x, y \in (\Sigma_{bool})^*$:

Wenn $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$, dann gilt für alle $z \in (\Sigma_{bool})^*$:

$$xz \in L(B_k) \iff yz \in L(B_k)$$

Die Idee des Beweises ist es, eine Menge S_k von Wörtern zu finden, so dass für keine zwei unterschiedlichen Wörter $x, y \in S_k$ die Gleichung $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$ gelten darf. Dann müsste B_k mindestens $|S_k|$ viele Zustände haben.

Wir wählen $S_k = (\Sigma_{bool})^k$ und zeigen, dass $\hat{\delta}_k(q_{0k}, q)$ paarweise unterschiedliche Zustände für alle $u \in S_k$ sind.

Wir beweisen dies per Widerspruch.

Seien $x = x_1x_2\dots x_k$ und $y = y_1y_2\dots y_k$ für $x_i, y_i \in \Sigma_{bool}, i \in \{1, \dots, k\}$ zwei unterschiedliche Wörter aus S_k .

Nehmen wir zum Widerspruch an, dass $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$.

Weil $x \neq y$, existiert ein $j \in \{1, \dots, k\}$, so dass $x_j \neq y_j$. O.B.d.A. setzen wir $x_j = 1$ und $y_j = 0$. Betrachten wir nun $z = 0^{j-1}$. Dann ist

$$xz = x_1\dots x_{j-1}1x_{j+1}\dots x_k0^{j-1} \text{ und } yz = y_1\dots y_{j-1}0y_{j+1}\dots y_k0^{j-1}$$

und daher $xz \in L_k$ und $yz \notin L_k$. Dies ist ein Widerspruch! Folglich gilt $\hat{\delta}_k(q_{0k}, x) \neq \hat{\delta}_k(q_{0k}, y)$ für alle paarweise unterschiedliche $x, y \in S_k = (\Sigma_{bool})^k$.

Daher hat B_k mindestens $|S_k| = 2^k$ viele Zustände. ■

Kapitel 4

Lemma 4.2

Für jede Mehrband-TM A existiert eine zu A äquivalente TM B .

Beweis:

Sei A eine k -Band-Turingmaschine für ein $k \in \mathbb{N} \setminus \{0\}$. Wir konstruieren eine TM B , die Schritt für Schritt A simuliert.

B speichert die Inhalte aller $k + 1$ Bänder von A auf ihrem einzigen Band. Anschaulich gesprochen ist jedes Feld auf dem Band von B ein $2(k + 1)$ -Tupel und jedes Element dieses Tupels ist auf einer Spur. Sei Γ_A das Arbeitsalphabet von A . Dann gilt

$$\Gamma_B = (\Sigma_A \cup \{\$, \pounds, \sqcup\}) \times \{\sqcup, \uparrow\} \times (\Gamma_A \times \{\sqcup, \uparrow\})^k \cup \Sigma_A \cup \{\$, \pounds\}$$

Für ein Symbol $\alpha = (a_0, a_1, a_2, \dots, a_{2k+1}) \in \Gamma_B$ sagen wir, dass a_i auf der i -ten Spur liegt. Daher bestimmen die i -ten Elemente der Symbole auf dem Band von B den Inhalt der i -ten Spur. Eine Konfiguration $(q, w, i, x_1, i_1, x_2, i_2, \dots, x_k, i_k)$ von A ist dann in B wie folgt gespeichert.

- Der Zustand q ist in der endlichen Kontrolle von B gespeichert.
- Die 0-te Spur des Bandes von B enthält die $\pounds w \$$ (i.e. den Inhalt des Eingabebandes von A)
- Für alle $i \in \{1, \dots, k\}$ enthält die $(2i)$ -te Spur des Bandes von B den Inhalt vom i -ten Band von A (i.e. $\pounds x_i \$$).
- Für alle $i \in \{1, \dots, k\}$ bestimmt die $(2i + 1)$ -te Spur des Bandes von B mit dem Symbol \uparrow die Position des Kopfes auf dem i -ten Arbeitsband von A .

Ein Schritt von A kann jetzt durch folgende Prozedur von B simuliert werden:

1. B liest einmal den Inhalt ihres Bandes von links nach rechts, bis sie alle $k + 1$ Kopfpositionen von A gefunden hat, und speichert dabei in ihrem Zustand die $k + 1$ Symbole, die an diesen Positionen stehen. (Dies kann ohne weiteres in der Zustandsmenge abgespeichert werden, da k fix ist, folglich ist dann Γ_A^k auch endlich)
2. Nach der ersten Phase kennt B das ganze Argument (der Zustand von A ist im Zustand von B gespeichert) der Transitionsfunktion von A und kann also die entsprechenden Aktionen (Köpfe bewegen, Ersetzen von Symbolen) von A bestimmen. Diese Änderungen führt B in einem Lauf über ihr Band von rechts nach links durch.

■

Kapitel 5

Satz 5.4

$\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.

Beweis:

Wir definieren eine injektive Funktion von $f : [0, 1] \rightarrow \mathcal{P}((\Sigma_{bool})^*)$ und beweisen so $|\mathcal{P}((\Sigma_{bool})^*)| \geq |[0, 1]|$.

Sei $a \in [0, 1]$ beliebig. Wir können a wie folgt binär darstellen: $\text{Nummer}(a) = 0.a_1a_2a_3a_4\dots$ mit $a = \sum_{i=1}^{\infty} a_i \cdot 2^{-i}$. Hier ist zu beachten, dass wir für eine Zahl a immer die lexikographisch letzte Darstellung. Dies tun wir, weil eine reelle Zahl 2 verschiedene Binärdarstellungen haben kann. Beispiel: $\frac{1}{2} = 0.1\bar{0} = 0.0\bar{1}$.

Für jedes a definieren wir:

$$f(a) = \{a_1, a_2a_3, a_4a_5a_6, \dots, a_{\binom{n}{2}+1}a_{\binom{n}{2}+2}\dots a_{\binom{n+1}{2}}, \dots\}$$

Da $f(a) \subseteq (\Sigma_{bool})^*$ gilt $f(a) \in \mathcal{P}((\Sigma_{bool})^*)$.

Wir haben für alle $n \in \mathbb{N} \setminus \{0\}$, dass $f(a)$ **genau** ein Wort dieser Länge enthält. Nun können wir daraus folgendes schliessen:

Weil die Binärdarstellung zweier unterschiedlichen reellen Zahlen an mindestens einer Stelle unterschiedlich ist, gilt $b \neq c \implies f(b) \neq f(c), \forall b, c \in [0, 1]$.

Folglich ist f injektiv und wir haben $|\mathcal{P}((\Sigma_{bool})^*)| \geq |[0, 1]|$.

Da $[0, 1]$ nicht abzählbar ist, folgt daraus:

$\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.

■

Satz 5.5

$L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$.

Beweis:

Wir haben

$$L_{\text{diag}} = \{w \mid w = w_i \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht für ein } i \in \mathbb{N} \setminus \{0\}\}$$

Widerspruchsbeweis:

Sei $L_{\text{diag}} \in \mathcal{L}_{\text{RE}}$. Dann existiert eine TM M , so dass $L(M) = L_{\text{diag}}$. Da diese TM eine TM in der Nummerierung aller TM ist, existiert ein $i \in \mathbb{N}$, so dass $M_i = M$.

Wir betrachten nun das Wort w_i für diese $i \in \mathbb{N}$. Per Definition von L_{diag} , gilt:

$$w_i \in L_{\text{diag}} \iff w_i \notin L(M_i)$$

Da aber $L(M_i) = L_{\text{diag}}$, haben wir folgenden Widerspruch:

$$w_i \in L_{\text{diag}} \iff w_i \notin L_{\text{diag}}$$

Folglich gilt $L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$. ■

Lemma 5.4

Sei Σ ein Alphabet. Für jede Sprache $L \subseteq \Sigma^*$ gilt:

$$L \leq_R L^c \text{ und } L^c \leq_R L$$

Beweis: Es reicht $L^c \leq_R L$ zu zeigen, da $(L^c)^c = L$ und somit dann $(L^c)^c = L \leq_R L^c$.

Sei M' ein Algorithmus für L , der immer hält ($L \in \mathcal{L}_R$). Dann beschreiben wir einen Algorithmus B , der L^c entscheidet.

B übernimmt die Eingaben und gibt sie an M' weiter und invertiert dann die Entscheidung von M' . Weil M' immer hält, hält auch B immer und wir haben offensichtlich $L(B) = L^c$. ■

Korollar 5.2 (bzw. Anwendung von Lemma 5.4)

$(L_{\text{diag}})^c \notin \mathcal{L}_R$.

Beweis:

Aus Lemma 5.4 haben wir $L_{\text{diag}} \leq_R (L_{\text{diag}})^{\mathbb{C}}$. Daraus folgt $L_{\text{diag}} \notin \mathcal{L}_R \implies (L_{\text{diag}})^{\mathbb{C}} \notin \mathcal{L}_R$. Da $L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$ gilt auch $L_{\text{diag}} \notin \mathcal{L}_R$.

Folglich gilt $(L_{\text{diag}})^{\mathbb{C}} \notin \mathcal{L}_R$. ■

Lemma 5.8

$L_{H,\lambda} \notin \mathcal{L}_R$.

Beweis:

Wir zeigen $L_H \leq_{\text{EE}} L_{H,\lambda}$. Wir beschreiben einen Algorithmus B , so dass $x \in L_H \iff B(x) \in L_{H,\lambda}$.

Für jede Eingabe arbeitet B wie folgt:

- Falls x von der falschen Form, dann $B(x) = M_{\text{inf}}$, wobei M_{inf} unabhängig von der Eingabe immer unendlich läuft.
- Sonst $x = \text{Kod}(M)\#w$: Dann $B(x) = M'$, wobei M' die Eingabe ignoriert und immer M auf w simuliert.

Wir sehen, dass M' genau dann auf λ hält, wenn $x \in L_H$.

Daraus folgt $x \in L_H \iff B(x) \in L_{H,\lambda}$. ■

Kapitel 6**Lemma 6.1**

Sei k eine positive ganze Zahl. Für jede k -Band Turingmaschine A , die immer hält, existiert eine äquivalente 1-Band-TM B , so dass

$$\text{Space}_B(n) \leq \text{Space}_A(n)$$

Beweisskizze:

Gleiche Konstruktion wie in Lemma 4.2. Wir können leicht sehen, dass B genau so viele Felder braucht, wie A .

Lemma 6.2

Zu jeder MTM A existiert eine äquivalente MTM B mit

$$\text{Space}_B(n) \leq \frac{\text{Space}_A(n)}{2} + 2$$

Beweisskizze:

Wir fassen jeweils 2 Felder von A zu einem Feld in B zusammen. $\Gamma_B = \Gamma_A \times \Gamma_A$. Wir addieren 1 für das \dagger am linken Rand und 1 für das Aufrunden im Fall von ungerader Länge.

Lemma 6.3

$\text{TIME}(t) \subseteq \text{SPACE}(t)$

Beweisskizze: In t Schritten sind höchstens t Felder beschreibbar.

Lemma 6.4

Sei S platzkonstruierbar. Für jede MTM M , für welche $\text{Space}_M(w) \leq s(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert eine äquivalente MTM M' , welche dies für alle $w \in \Sigma^*$ erfüllt.

Beweisskizze: Erzeuge für jede Eingabe $x \in \Sigma^*$ zuerst $0^{s(|x|)}$ auf einem zusätzlichen Band und nutze das als Platzüberwachung. Wenn M' diesen Platz überschreiten will, wird die Simulation unterbrochen und die Eingabe verworfen.

Lemma 6.5

Sei t zeitkonstruierbar. Zu jeder MTM, welche $\text{Time}_M(w) \leq t(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert eine äquivalente MTM M' , welche zumindest $\text{Time}_M(w) \leq 2t(|w|) \in \mathcal{O}(t(|w|))$ für alle $w \in \Sigma^*$ erfüllt.

Beweisskizze: Schreibe für jede Eingabe $x \in \Sigma^*$ $0^{t(|x|)}$ auf ein zusätzliches Arbeitsband und nutze dies zur Zeitählung. Wenn M' mehr Schritte machen will, wird die Simulation abgebrochen und die Eingabe verworfen.

Satz 6.2

Für jede Funktion s mit $s(n) \geq \log_2(n)$ gilt:

$$\text{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{s(n)})$$

Beweis:

Sei $L \in \mathbf{SPACE}(s(n))$. Nach Lemma 6.1 existiert eine 1-Band-TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, die **immer hält**, so dass $L = L(M)$ und $\text{Space}_M(n) \leq d \cdot s(n)$ für $d \in \mathbb{N}$ gelten. Für jede Konfiguration $C = (q, w, i, x, j)$ von M definieren wir die **innere Konfiguration von C** als

$$\text{In}(C) = (q, i, x, j).$$

Die innere Konfiguration enthält das Eingabewort w nicht, da dies sich während einer Berechnung nicht ändert.

Wir betrachten die Menge aller inneren Konfigurationen, dass bei einer **deterministischen** TM jede Berechnung $D = C_1, C_2, C_3, \dots$ von M auf einem Wort w mit $|w| = n$, die länger als

1 EE-Reduktionen und R-Reduktionen – Komplexitätsbeweise

Mit Inspiration von der Zsf. von Fabian Frei

Generelle Bemerkungen:

- L rekursiv (entscheidbar) $\iff L \in \mathcal{L}_R$
- L rekursiv aufzählbar $\iff L \in \mathcal{L}_{RE}$
- "Algorithmus" ist ein anderes Wort für eine Turingmaschine, die **immer** terminiert.

1.1 $L \in \mathcal{L}_R$

Wir kennen zwei Methoden um dies zu beweisen:

- Wir finden eine Sprache $L' \in \mathcal{L}_R$ und zeigen $L \leq_R L'$. (Meistens ein wenig umständlich)
- Direkter Beweis: Eine TM (bzw. ein Algorithmus) A beschreiben, so dass $L(A) = L$ und A immer terminiert.

1.2 $L \notin \mathcal{L}_R$

Wir kennen hier auch 3 Arten:

- Folgt sofort aus $L \notin \mathcal{L}_{RE}$, da $\mathcal{L}_R \subset \mathcal{L}_{RE}$.
- Wir wählen eine Sprache L' , so dass $L' \notin \mathcal{L}_R$ und beweisen $L' \leq_{R/EE} L$.
Geeignete Sprachen als L' sind: $L_{\text{empty}}^{\mathcal{L}}, L_{\text{diag}}^{\mathcal{L}}, L_H, L_U, L_{H,\lambda}$. (Alle im Buch bewiesen)
- Satz von Rice

Für den **Satz von Rice**:

- Wir können mit diesem Satz nur $L \notin \mathcal{L}_R$ beweisen!
- Wir haben folgende Bedingungen:
 1. $L \subseteq \text{KodTM}$
 2. $\exists \text{ TM } M: \text{Kod}(M) \in L$
 3. $\exists \text{ TM } M: \text{Kod}(M) \notin L$
 4. $\forall \text{ TM } M_1, M_2: L(M_1) = L(M_2) \implies (\text{Kod}(M_1) \in L \iff \text{Kod}(M_2) \in L)$

Für den letzten Punkt (4) muss man überprüfen, ob in der Definition von $L = \{\text{Kod}(M) \mid M \text{ ist TM und } \dots\}$ überall nur $L(M)$ vorkommt und nirgends M direkt. Beziehungsweise reicht es, wenn man die Bedingung so umschreiben kann, dass sie nur noch durch $L(M)$ beschrieben ist.

1.3 $L \in \mathcal{L}_{RE}$

Wir beschreiben eine TM M mit $L(M) = L$, die nicht immer halten muss.

Meistens muss die TM eine Eigenschaft, für alle möglichen Wörter prüfen. (Bsp: $\text{Kod}(M_1) \in L_H^C$: Wir gehen alle Wörter durch, um dasjenige zu finden, für das M_1 hält.)

Wir verwenden oft einen von den folgenden 2 Tricks, um dies zu tun:

- Da es für jede NTM M' , eine TM M gibt, so dass $L(M') = L(M)$, können wir eine solche definieren, für die $L(M') = L$ gilt.
- Die andere Variante, ist die parallele Simulation von Wörtern, bei dem man das Diagonalisierungsverfahren aus dem Buch verwendet. (Bsp: Beweis $L_{\text{empty}} \in \mathcal{L}_{RE}$, S. 156 Buch)

1.4 $L \notin \mathcal{L}_{RE}$

Hier haben wir 2 mögliche (offizielle) Methoden:

- Diagonalisierungsargument mit Widerspruch, wie beim Beweis von $L_{\text{diag}} \notin \mathcal{L}_{RE}$.
- Widerspruchsbeweis mit der Aussage $L \in \mathcal{L}_{RE} \wedge L^C \in \mathcal{L}_{RE} \implies L \in \mathcal{L}_R$.

Inoffiziell könnten wir auch die EE-Reduktion verwenden, wird aber weder in der Vorlesung noch im Buch erwähnt.

1.5 EE- und R-Reduktionen: Tipps und Tricks

- Die vorgeschaltete TM A muss immer terminieren! I.e. sie muss ein Algorithmus sein.
- Die Eingabe sollte immer zuerst auf die Richtige Form überprüft werden!
Auch im Korrektheitsbeweis, sollte dieser Fall als erstes abgehandelt werden.
- Für Korrektheit müssen wir immer $x \in L_1 \iff A(x) \in L_2$ beweisen.
- Wir verwenden meistens folgende 2 Tricks:
 1. Transitionen nach q_{accept} oder q_{reject} umleiten nach q_{reject}/q_{accept} oder einer **Endlosschleife**.
 2. TM M' konstruieren, die ihre Eingabe ignoriert und immer dasselbe tut (z.B. eine TM dessen Kodierung gegeben ist, auf ein fixes Wort simulieren).
- Die Kodierung einer TM generieren, dessen Sprache gewisse Eigenschaften hat (z.B. sie akzeptiert alle Eingaben, läuft immer unendlich etc.)

2 Polynomialzeitreduktionen

Typische Aufgabe: L ist NP-Vollständig. Dann müssen wir (i) L in NP und (ii) L ist NP-schwer zeigen.

- Wir beschreiben eine NTM M , so dass $L(M) = L$. M errät (nichtdeterministisch) ein Zertifikat und verifiziert dies (deterministisch) in Polynomialzeit. M akzeptiert, wenn die Verifikation erfolgreich ist.
 M akzeptiert $\iff M$ hat eine akzeptierende Berechnung
- Wir nehmen eine Sprache L' die NP-Schwer ist und zeigen $L' \leq_p L$.
Beweisidee:
Wir zeigen eine Reduktion indem wir einen Polynomialzeit Algorithmus A beschreiben, so dass $x \in L \iff A(x) \in L'$. Wir müssen also folgende 2 Punkte für A beweisen:
 - $x \in L \iff A(x) \in L'$ (meist recht komplex, beide Richtungen einzeln beweisen)
 - A läuft in Polynomialzeit (meist trivial, es reicht eine High-Level Begründung zu geben)
 - Wir könnten es auch direkt beweisen (wie Beweis vom Satz von Cook). Dies ist aber meist zu komplex.

3 Grammatiken

Beispiel 10.6

Sei $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

Beweis durch Widerspruch:

Sei L kontextfrei. Dann gilt das Pumping Lemma für kontextfreie Sprachen.

Sei n_L die Konstante aus dem Pumping Lemma.

Dann wählen wir $z = a^{n_L} b^{n_L} c^{n_L}$, $|z| \geq n_L$, $z \in L$.

Dann gilt für jede Partition $z = uvwxy$ mit (i) $|vx| \geq 1$ und (ii) $|vwx| \leq n_L$, auch (iii) $\{uv^iwx^iy \mid i \in \mathbb{N}\}$.